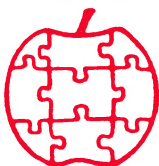


Apple

\$1.80



Assembly Line

Volume 5 -- Issue 11

August, 1985

In This Issue...

//c + Z-RAM = 576K Printer Buffer	2
How Many Bytes for Each Opcode?	12
Generic Conversion Routines.	17
Wildcard Filename Search	22

The 65816 continues to make news. We hear of at least two major books on 65816 Assembly Language, which should be in print soon. We also hear that sales of the chip are taking off, with some firms ordering multiplied thousands. Although we have yet to SEE one, we keep hearing reports of plug-in boards for Apples that contain a 65816 and lots of RAM: ComLog, MicroMagic, Checkmate Technology, and others.

Meanwhile, we contemplate the future advantages to just enhancing existing Apples with 65802's and big RAM boards. Applied Engineering or Checkmate will be delighted to stuff 512K additional RAM into your //c. You can add five times that much to your //e with AE's latest version of RAMWorks. Apple's forthcoming Slinky card will add up to a megabyte to any II, II Plus, or //e with a spare slot (1-7). Call APPLE's latest magazine offers the BIG BOARD for slot 0-7 use, one megabyte addressable either in Slinky fashion or with "standard" D000-FFFF mapping, for only \$599. If you hurry, they have a special (even lower) price good until Sept 30th.

6800 Cross Assembler for ProDOS

The S-C 6800 Macro Cross Assembler is now also available in a ProDOS version. This is the Version 2.0 level Cross Assembler, including the additional opcodes of the Motorola 6801 and Hitachi 6301 microprocessors. Either the DOS or the ProDOS Version 2.0 Cross Assembler is \$50; if you already have one you can add the other for only \$20.

//c + Z-RAM = 576K Printer Buffer.....David C. Johnson
Applied Engineering

What has 640K of memory and is as cute as a button? My Apple //c! It didn't come with all that memory, "only" 128K of it. Before I even powered it up for the first time, I installed a 512K Z-RAM. Ready to take on Blue's 640K machine? Maybe.

I've had quite a few Apple Computers, my first had Integer ROMs and a serial number in the thirty one thousands, and my current workhorse is an Apple //e with the works. So why a //c? Well, for one it's cute, and secondly its firmware was written by Ernie Beernink and Rich Williams, the same guys that wrote the //e Enhanced ROMs and Extended Debugging Monitor. These guys write slick code. Finally, I can type control-reset with one hand.

Well, what to do after getting it home? I tried my mouse out on it, but moved it back to the //e. My paddles and joysticks all have 16-pin plugs, so I couldn't use them. I don't have an RGB interface for the //c yet, so the color monitor has to stay put. That leaves my Imagewriter printer to play with.

Having two computers and only one printer is an old problem. One usually solved with a rotary switch. I figured that I could do a little better. What I did is connect the Imagewriter to the //c's Printer port, and the //e's Super Serial Card (SSC) to the //c's Modem port. I then wrote the program that follows this article. It implements a 576K buffer for the //e, in the //c. Now I can use the printer from the //c just by typing pr#1. When I want to print from the //e, I just boot a disk on the //c, then type pr#1 on the //e. However, the printing, for the //e, goes MUCH faster. I've setup the link between the //e and the //c to transmit at 19200 baud! Assembling a listing of the buffering program takes about 7 seconds (and half of that is writing the target file)!

The SSC is in slot 1, it is configured as follows:

SW1: off off off off off on on
SW2: on off off on on off off
The jumper block is installed pointing towards modem

The Imagewriter's switches are set:

SW1: open open open open closed closed open open
SW2: closed closed open open.

The pieces are connected with two DIN 5-Pin(m) to DB-25(m) cables, Apple Model Number: A9C0308 (4-2, 2-3, 1-6, 3-7, and 5-20). The cable from the //e to the //c is plugged into a //c System Clock which in turn is plugged into the Modem Port.

The program should work with most any serial printer, and serial card, however, if the serial card cannot "eliminate the modem", you will need a modem-eliminator cable extension, or will have to reverse pins 2 and 3 and pins 6 and 20 of the DB-25 connector. The Apple cable I used cannot be modified.

S-C Macro Assembler Version 2.0.....DOS \$100, ProDOS \$100, both for \$120
 ProDOS Upgrade Kit for Version 2.0 DOS owners.....\$30
 Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.....\$20
 Full Screen Editor for S-C Macro (with complete source code).....\$49
 S-C Cross Reference Utility.....without source code \$20, with source \$50
 RAK-Ware DISASM.....\$30
 Source Code for DISASM.....additional \$30
 S-C Word Processor (with complete source code).....\$50
 DP18 Source and Object.....\$50
 Double Precision Floating Point for Applesoft (with source code).....\$50
 "Bag of Tricks", Worth & Lechner, with diskette.....(\$39.95) \$36
 MacASM -- Macro Assembler for Macintosh (Mainstay).....(\$150.00) \$100
 S-C Documentor (complete commented source code of Applesoft ROMs).....\$50
 Source Code of //e CX & F8 ROMs on disk.....\$15
 Cross Assemblers for owners of S-C Macro Assembler.....\$32.50 to \$50 each

(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, G1650/70, others)

AAL Quarterly Disks.....each \$15, or any four for \$45

	1980	Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
Each disk contains	1980	-	-	-	1
the source code from	1981	2	3	4	5
three issues of AAL,	1982	6	7	8	9
saving you lots of	1983	10	11	12	13
typing and testing.	1984	14	15	16	17
	1985	18	19		

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Verbatim Diskettes (with hub rings)..... package of 20 for \$32
 Vinyl disk pages, 6"x8.5", hold two disks each.....10 for \$6
 Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
 (Cardboard folders designed to fit 6"x9" Envelopes.) or \$25 per 100
 Envelopes for Diskette Mailers..... 6 cents each

65802 Microprocessor (Western Design Center).....(\$95) \$50
 quikLoader EPROM System (SCRG).....(\$179) \$170
 PROMGRAMMER (SCRG).....(\$149.50) \$140
 Switch-a-Slot (SCRG).....(\$190) \$175
 Extend-a-Slot (SCRG).....(\$35) \$32

"Apple ProDOS: Advanced Features for programmers", Little..(\$17.95) \$17
 "Inside the Apple //c", Little.....(\$19.95) \$18
 "Inside the Apple //e", Little.....(\$19.95) \$18
 "Apple II+/IIf Troubleshooting & Repair Guide", Brenner.....(\$19.95) \$18
 "Apple II Circuit Description", Gayler.....(\$22.95) \$21
 "Understanding the Apple II", Sather.....(\$22.95) \$21
 "Understanding the Apple //e", Sather.....(\$24.95) \$23
 "Enhancing Your Apple II, vol. 1", Lancaster.....(\$15.95) \$15
 "Enhancing Your Apple II, vol. 2", Lancaster.....(\$17.95) \$17
 "Assembly Cookbook for the Apple II/IIf", Lancaster.....(\$21.95) \$20
 "Beneath Apple DOS", Worth & Lechner.....(\$19.95) \$18
 "Beneath Apple ProDOS", Worth & Lechner.....(\$19.95) \$18
 "6502 Assembly Language Programming", Leventhal.....(\$18.95) \$18
 "6502 Subroutines", Leventhal.....(\$18.95) \$18
 "Real Time Programming -- Neglected Topics", Foster.....(\$9.95) \$9
 "Microcomputer Graphics", Myers.....(\$12.95) \$12
 "Assem. Language for Applesoft Programmers", Finley & Myers.(\$16.95) \$16
 "Assembly Lines -- the Book", Wagner.....(\$19.95) \$18
 "AppleVisions", Bishop & Grossberger.....(\$39.95) \$36

Add \$1.50 per book for US shipping. Foreign orders add postage needed.
 Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
 *** (214) 324-2050 ***
 *** We accept Master Card, VISA and American Express ***

While the listing included with this article requires a 512K Applied Engineering Z-RAM board, I have also written versions that work in a 256K Z-RAM and in a stock Apple //c. More on these versions later. The memory on a Z-RAM is implemented as additional banks of auxiliary memory. Which of the auxiliary banks is the current auxiliary bank is controlled by a new hardware location at \$C073. The Z-RAM powers-up disabled, that is, with the //c's built-in auxiliary bank as the current auxiliary bank. The //c powers-up with main memory enabled and all auxiliary memory disabled. Once selected as the current auxiliary bank, a Z-RAM bank is switched around by all the normal soft switches in the same manner as the //c's built-in auxiliary bank. A 512K Z-RAM has 8 additional banks and a 256K Z-RAM has 4 more. Which additional bank is the current auxiliary bank is selected by writing an ODD number between 1 and \$F (inclusive) to the bank register at \$C073. The 4 most significant data bits are ignored and any even number (usually zero) selects the //c's built-in auxiliary bank. A 256K Z-RAM only has bank numbers 3, 7, \$B, and \$F. To ease the task of writing programs that display 80 columns of text or double hires graphics, video data is always fetched from the //c's banks, even if a Z-RAM bank is the current auxiliary bank. Because the Z-RAM plugs into the processor and MMU sockets of the //c, and since only one board may be added this way, the Z-RAM includes a Z-80 processor. The Z-RAM is also totally compatible with the RamWorks board for the //e.

The //c's serial ports are a lot like Super Serial Cards in slots 1 and 2 of a //e. The ports and the SSC both use the 6551 ACIA (Asynchronous Communications Interface Adapter) and the firmware is quite similar. There is one significant difference that I found. The SSC tells an external source of data to stop transmitting by asserting the Data Terminal Ready bit of the ACIA command register (and thus the DTR pin when the jumper block is in the terminal position), while the //c's ports control the DTR pin with the Request To Send (and transmitter control) bits. It's right there on page 254 of The Apple //c Reference Manual Volume 1. Compare this to the schematic on page 100 of the SSC Manual.

Because every //c has a 65C02 processor, I can write code using the new opcodes and it will work in other peoples' machines. Of course if the code will also work in a //e, I can not be sure that it will be executed on a 65C02. With the release of the //e enhancement kit, this situation should improve. 65802 opcodes, being new and rare, must be reserved for programs intended for use in a very few machines.

On to the program. The target file is intended to load at \$2000 in main memory. The code from lines 32 to 73 is executed in the \$2000 area. This section does all of the setup for what is to come. The D and I flags are cleared and set respectively, ten soft switches are thrown, the screen is cleared, the remainder of the code is copied into ALL auxiliary zero pages and stacks, a text message is written to the screen, and the two ACIAs are initialized. The code copy and message printing share a loop. Lines 66 and 70 cheat a little. The

INCs are assembled and the LDA #s are treated as comments. They work because the would-be operands of the LDA #s are one greater than the values just loaded by the previous LDA #s. The 'A' in line 74 is an open-apple MouseText character. The code in aux bank 0 is then entered at label 'Scan'.

The routines 'Write' and 'Read' (lines 79 and 88), handle all access to the buffer. In 'Write', the aux bank is selected, the address within that bank is written into the operand of a store absolute instruction (the copy in the bank just selected), and then the data byte is written. That's a total of four bytes of information passed in internal registers. The data byte had to be passed in the stack pointer! It couldn't have been passed in a memory location because it would have been switched out. 'Read' is a little simpler, it returns a data byte in the Acc. Since I'm using the S-reg for data and the aux bank 0 stack page for code, the program doesn't make any use of regular stack operations. After re-selecting aux bank 0, 'Write' and 'Read' jump back to the code just after the jumps that 'called' them. Even though the \$2000 code copied the entire image into every aux bank, only 'Write' and 'Read' are not used as buffer in the Z-RAM banks.

Lines 99 to 108 allocate the (zero page!) variables required to keep track of the buffer. The 'Receive' variables indicate where the next byte received will be buffered, the 'Transmit' variables indicate where the next byte to be printed is buffered, and the 'Byte.Counter' variables keep track of how full (or empty) the buffer is. If the byte counter is zero, then the 'Transmit' variables are equal to the 'Receive' variables and the buffer is empty. 'RTS.Bit' is used to keep track of the //c's 'select' state.

Lines 110 to 128 run an indicator at the top-center of the screen and check to see if you've pressed a key. If you press the space bar, and if the program hasn't asserted the Request To (NOT) Send bit (because the buffer is nearly full), the //e may be halted. This works like a printer's select button.

Lines 129 to 207 handle buffering incoming data. If the Modem ACIA detects any transmission errors, you will see an indication of this at the left end of screen line three. If no character has been received, we go check the Printer port. When a character has been received, we test if the buffer is almost full. If it is, we assert RTS' (another character may already be on the way). The byte counter is incremented. If the buffer is completely full, we tick the third position of screen line one and go check the Printer port. This means that the RTS' handshaking isn't working. You will also get overrun errors. If we have room for the character, we increment the upper left screen position, and load the character from the RxD reg into the stack pointer. We then load the 'Receive' variables, maybe juggle the address high order nibble for the overlapping language card banks, and call 'Write'. Upon return, the 'Receive' variables are advanced through the buffer memory, avoiding our program and invalid aux banks. We then fall into the Printer port code.

Lines 208 to 271 handle printing buffered data as the printer can take it. This code is similar to the code for incoming data. Fewer things can go wrong, we of course test for an empty TxD reg and an empty buffer. We check to see if the buffer is somewhat less than almost full, and may release RTS'. The byte counter is decremented here. When a character is to be printed, we increment the upper right screen position, load the 'Transmit' variables, maybe juggle, call 'Read' and stuff the character into the TxD reg. Upon return, the 'Transmit' variables are advanced (same way), and we loop to 'Scan'. Forever. Reset exits the program.

The program loops VERY quickly. It has to. At 19200 baud, a character is received from the //e every half millisecond and at 9600 baud, a character may be printed every millisecond. The pair of locations at the top center of the screen, that are changed every time around the loop, give a good indication of how fast things are happening. The locations in the upper corners (my //e is to the left of the //c and the printer is to the right) are a good representation of the values of the 'Receive' and 'Transmit' variables. When buffering, the receive indicator races ahead while the transmit indicator lags behind, but since they are both initialized to blanks and the appropriate one is incremented when a character is moved, they come to rest displaying the same character when the buffer is empty.

The symbols 'Z.RAM.Banks.Avail', 'Z.RAM.Banks.Used', 'IIC.Aux.Bank.Avail' and 'BufLen' (lines 94, 96, 273-274) determine the size of the buffer. The ADC immediate operands in lines 195 and 259 cause the buffer to advance from bank 0 to 1 to 3 to 5... to \$F. The listing is setup to use a //c's aux bank and a 512K Z-RAM. The changes for a 256K Z-RAM are easy: change the SAVE and .tf filenames (320K), change the 8 in line 96 to a 4, change the 9 in line 274 to a 5, and change the ADC #1s in lines 195 and 259 to ADC #3s. The changes for operating without a Z-RAM are not as simple. I removed all the bank stuff, made the byte counter only 16 bits, and combined the code copy with the screen clear instead of the message printing. It took about 5 minutes. The resulting code just fit into the aux zero page! The source code for all three versions will be on S-C Software's next quarterly disk, and I will send a paper listing of the //c only version to anyone who sends a self addressed stamped envelope to me care of Applied Engineering. I sometimes use the //c only version even though I have a Z-RAM. With the ProDrive disk emulation software, I can lock-out bank 0, leaving it available for double hires or a 64K buffer for my //e. With a 512K Z-RAM, I get a 1024 block /RAM volume.

The program does not use any main memory for the buffer because when you have 576K of aux memory, why bother programming for "only" another 64K? The //c only version, with 64K of buffer memory, is as big or bigger than most buffer boards/boxes. If anyone writes a 128K main/aux version of the program I would appreciate a copy.

NEW DON LANCASTER RELEASES

Available ONLY from Synergetics. All software open and unlocked.

ABSOLUTE RESET (IIC/old IIE/new IIE)\$19.50

Get back in total control. No more hole blasting! Access any part of any program at any time for any reason. Passes all diagnostics. Invisible till activated. Includes EPROM burner adapter details, service listings, and a free bonus book.

APPLEWRITER™ TOOLKITS (DOS 3.3 or ProDOS)\$39.50

EIGHT diskette sides include a complete disassembly script; source code capturing; self-prompting glossaries; Diablo microjustify and proportional space, patches including NULL, shortline, Grappler, IIC detaching, many others; answers to most help line questions; two columns; WPL secrets; space on disk; keyword indexing; multi and even-odd headers; bunches more.

Both TOOLKITS (sixteen disk sides)\$59.50

APPLEWORKS™ DISASSEMBLY SCRIPT\$49.50

TEN diskette sides chock full of Applework's innermost secrets, using Don's unique and powerful "tearing method". Primarily for gonzo hackers and definitely NOT for the faint of heart.

LASERWRITER/APPLEWRITER UTILITIES\$39.50

Two volume package gives you unmatchably superb IIE text and graphics. Included are automatic formatters, boxes and fancy borders, daisywheel changers, envelope and label routines, unbelievably fast formletters, grids and rulers, HIRES converters, DC1 patches, demos, justify routines, self-prompting glossaries, help screens, a sign shop, outlined text, more.

AUTOGRAPHED LANCASTER BOOKS:

Apple IIE Assembly Cookbook	\$21.50
All About Applewriter IIE	\$14.50
Enhancing Your Apple II & IIE, (Volume I or II)	\$15.50
Micro Cookbook (Volume I or II)	\$15.50
CMOS Cookbook	\$14.50
TTL Cookbook	\$12.50
Incredible Secret Money Machine	\$7.50
Complete book and software list	(FREE)

COMPANION DISKETTES:

For Enhancing Your Apple II, Volume I	\$19.50
For Enhancing Your Apple II, Volume II	\$19.50
For Don Lancaster's Assembly Cookbook	\$19.50

SYNERGETICS
746 First Street
Box 809-SC
Thatcher, AZ, 85552

FREE
VOICE HELPLINE
(602) 428-4073

Appleworks, Applewriter, and ProDOS are registered trademarks of Apple Computer. VISA and MASTERCHARGE accepted. Please - no COD, foreign, or purchase orders.

```

0001 ;SAVE Buf.576K
0002 -----
0003 Dedicated to Allan B. Calhamer.
0004 -----
C098- 0005 Printer.ACIA.TxD .eq $C098 (w)
C099- 0006 Printer.ACIA.Status .eq $C099 (r)
C09A- 0007 Printer.ACIA.Command .eq $C09A (r/w)
C09B- 0008 Printer.ACIA.Control .eq $C09B (r/w)
C0A8- 0009 Modem.ACIA.RxD .eq $C0A8 (r)
C0A9- 0010 Modem.ACIA.Status .eq $C0A9 (r)
C0AA- 0011 Modem.ACIA.Command .eq $C0AA (r/w)
C0AB- 0012 Modem.ACIA.Control .eq $C0AB (r/w)
C073- 0013 Z.RAM.Bank.Reg .eq $C073 (w) same as RamWorks
C000- 0014 Keyboard .eq $C000 (r)
C001- 0015 Store80 .eq $C001 (w) on
C003- 0016 RAMRd .eq $C003 (w) aux
C005- 0017 RAMWrt .eq $C005 (w) aux
C009- 0018 AltZP .eq $C009 (w) aux
C00C- 0019 Vid40 .eq $C00C (w)
C00F- 0020 SetAltChr .eq $C00F (w) w/MouseText
C010- 0021 Clear.Key.Strobe .eq $C010 (r)
C051- 0022 Text .eq $C051 (r)
C054- 0023 Page1 .eq $C054 (r) main
C055- 0024 Page2 .eq $C055 (r) aux
C057- 0025 Hires .eq $C057 (r) $2000-$3FFF too...
C083- 0026 LCRAM2 .eq $C083 (r/w; write doesn't
C08B- 0027 LCRAM1 .eq $C08B change write enable)
0028 -----
0029 .op 65C02
0030 .or $2000
0031 .tf Bufit576K
2000- 0032 dcj CLD rqd (now)
2001- D8 0033 SEI close this can of worms...
2002- AD 83 C0 0034 LDA LCRAM2 1x...switches setup
2005- AD 51 C0 0035 LDA Text
2008- AD 54 C0 0036 LDA Page1
200B- AD 57 C0 0037 LDA Hires
200E- 9C 01 C0 0038 STZ Store80
2011- 9C 03 C0 0039 STZ RAMRd
2014- 9C 05 C0 0040 STZ RAMWrt
2017- 9C 09 C0 0041 STZ AltZP
201A- 9C 0F C0 0042 STZ SetAltChr
201D- 9C 0C C0 0043 STZ Vid40
2020- A9 A0 0044 LDA #"" clear 40 column screen
2022- A2 00 0045 LDX #0
2024- 9D 00 04 0046 .1 STA $400,X
2027- 9D 00 05 0047 STA $500,X
202A- 9D 00 06 0048 STA $600,X
202D- 9D 00 07 0049 STA $700,X
2030- E8 0050 INX
2031- D0 F1 0051 BNE .1
2033- A0 0F 0052 LDY #$0F install Image in aux ZPs/Stacks
2035- 8C 73 C0 0053 STY Z.RAM.Bank.Reg
2038- BD 77 20 0054 .2 LDA Image,X
203B- 95 00 0055 STA $00,X
203D- BD 77 21 0056 LDA Image+$100,X
2040- 9D 00 01 0057 STA $100,X
2043- E8 0058 INX
2044- D0 F2 0059 BNE .3
2046- B9 67 20 0060 LDA Msg,Y put up a message
2049- 99 0C 05 0061 STA $50C,Y
204C- 88 0062 DEY
204D- 10 E6 0063 BPL .2
204F- A9 0A 0064 LDA #$000.0.10.1.0 bop ACIAs
2051- 8D 9A C0 0065 STA Printer.ACIA.Command
2054- 1A 0066 inc LDA #$000.0.10.1.1 RTS' lo
2055- 8D AA C0 0067 STA Modem.ACIA.Command
2058- A9 1E 0068 LDA #$0.00.1.1110 9600 baud
205A- 8D 9B C0 0069 STA Printer.ACIA.Control
205D- 1A 0070 inc LDA #$0.00.1.1111 19200 baud!
205E- 8D AB C0 0071 STA Modem.ACIA.Control
2061- AD A8 C0 0072 LDA Modem.ACIA.RxD
2064- 4C 2B 00 0073 JMP Scan go 2 it
2067- 41 0074 Msg .AS 'A' as in Apple
2068- A0 AF AF
206B- E3 A0 E2
206E- F5 E6 E6
2071- E5 F2 A0
2074- F0 E7 ED 0075 .AS -" //c buffer pgm"

```



```

0076 Image .ph $00
0077 ; aux bank specified by Acc, bank adr lo by X-reg,
0078 ; bank adr hi by Y-reg, and byte passed in S-regl
0000- 8D 73 C0 0079 Write STA Z.RAM.Bank.Reg bank in Z-RAM
0003- 86 09 0080 STX <.1+1 modify STX operand in "this" bank
0005- 84 0A 0081 STY <.1+2
0007- BA 0A 0082 TSX get byte to a usable regl
0008- 8E FF FF 0083 .1 STX $FFFF abs adr modified for each write
000B- 9C 73 C0 0084 STZ Z.RAM.Bank.Reg revert to //c aux bank
000E- 4C D5 00 0085 JMP W.Ret
0086 ; aux bank specified by Acc, bank adr lo by X-reg,
0087 ; bank adr hi by Y-reg, and byte returned in Acc.
0011- 8D 73 C0 0088 Read STA Z.RAM.Bank.Reg bank in Z-RAM
0014- 86 19 0089 STX <.1+1 modify LDA operand in "this" bank
0016- 84 1A 0090 STY <.1+2
0018- AD FF FF 0091 .1 LDA $FFFF abs adr modified for each read
001B- 9C 73 C0 0092 STZ Z.RAM.Bank.Reg revert to //c aux bank
001E- 4C 55 01 0093 JMP R.Ret
1E- 0094 Z.RAM.Banks.Avail .eq #-3
0095 ; (-3 because JMP R.Ret never executed in Z-RAM)
F0- 0096 Z.RAM.used .eq Z.RAM.Banks.Avail*8
0097 -----
0098 ; buffer starts at first available location in //c aux bank
0021- 7F 0099 Receive.Adr.Lo .da #IIC.Aux.Bank.Avail
0022- 01 0100 Receive.Adr.Hi .da /IIC.Aux.Bank.Avail
0023- 00 0101 Receive.Bank .da #$00
0024- 7F 0102 Transmit.Adr.Lo .da #IIC.Aux.Bank.Avail
0025- 01 0103 Transmit.Adr.Hi .da /IIC.Aux.Bank.Avail
0026- 00 0104 Transmit.Bank .da #$00
0027- 00 0105 Byte.Counter.Lo .da #$000000 indicates empty
0028- 00 0106 Byte.Counter.Mid .da #$000000/256
0029- 00 0107 Byte.Counter.Hi .da #$000000/65536
002A- 08 0108 RTS.Bit .da #$000.0.10.0.0 RTS' lo
0109 -----
002B- AD 54 C0 0110 Scan LDA Page1 access main text screen
002E- EE 13 04 0111 INC #413 show we're alive
0031- CE 14 04 0112 DEC #414
0034- AD 55 C0 0113 LDA Page2 back to aux
0037- AD 00 C0 0114 LDA Keyboard scan keyboard
003A- 10 1E 0115 BPL Scan.Modem.Port
003C- C9 A0 0116 CMP # " space toggles RTS' (DTR2B) to //e
003E- D0 17 0117 BNE .2
0040- AD AA C0 0118 LDA Modem.ACIA.Command
0043- 29 08 0119 AND #$000.0.10.0.0
0045- D0 04 0120 BNE .1 =>It's ok, you can turn it off...
0047- A5 2A 0121 LDA RTS.Bit
0049- D0 0F 0122 BNE Scan.Modem.Port =>don't do it! (yet)
004B- AD AA C0 0123 .1 LDA Modem.ACIA.Command
004E- 49 08 0124 EOR #$000.0.10.0.0
0050- 8D AA C0 0125 STA Modem.ACIA.Command
0053- 29 08 0126 AND #$000.0.10.0.0
0055- 85 2A 0127 STA RTS.Bit
0057- 2C 10 C0 0128 .2 BIT Clear.Key.Strobe
0129 Scan.Modem.Port
005A- AC A9 C0 0130 LDY Modem.ACIA.Status
005D- 98 0131 TYA
005E- 29 07 0132 AND #$0000.0111 error bits mask
0060- F0 0A 0133 BEQ .1 =>error-free operation
0062- AA 0134 TAX
0063- AD 54 C0 0135 LDA Page1 access main text screen
0066- FE FF 04 0136 INC #4FF,X indicate error...
0069- AD 55 C0 0137 LDA Page2 back to aux
006C- 98 0138 .1 TYA
006D- 29 08 0139 AND #$0000.1000 receive data reg full mask
006F- F0 35 0140 BEQ CanRx =>not full
0071- A5 27 0141 LDA Byte.Counter.Lo received a byte,
0073- A6 28 0142 LDX Byte.Counter.Mid do we assert RTS' ?
0075- A4 29 0143 LDY Byte.Counter.Hi
0077- C9 91 0144 CMP #BufLen-256
0079- D0 0F 0145 BNE .2 =>buffer not @ full-256
007B- E0 FC 0146 CPX /BufLen-256
007D- D0 0B 0147 BNE .2 =>buffer not @ full-256
007F- C0 08 0148 CPY ^BufLen-256
0081- D0 07 0149 BNE .2 =>buffer not @ full-256
0083- A9 08 0150 LDA #$000.0.10.0.0 assert RTS'
0085- 1C AA C0 0151 TRB Modem.ACIA.Command
0088- A5 27 0152 LDA Byte.Counter.Lo reload it

```

```

008A- 1A      0153 .2    INC                      fig next byte count
008B- D0 04   0154      BNE .3
008D- E8      0155      INX
008E- D0 01   0156      BNE .3
0090- C8      0157      INY
0091- C9 91   0158 .3    CMP #BufLen      do we have room for it ?
0093- D0 13   0159      BNE Room          =>buffer not full
0095- E0 FD   0160      CPX /BufLen
0097- D0 0F   0161      BNE Room          =>buffer not full
0099- C0 08   0162      CPY BufLen
009B- D0 08   0163      BNE Room          =>buffer not full
009D- AD 54 C0 0164      LDA Page1        access main text screen
00A0- EE 02 04 0165      INC $402        indicate full
00A3- AD 55 C0 0166      LDA Page2        back to aux
00A6- 80 51   0167 CantRx BRA Cant.Receive =>buffer is full!
00A8- 85 27   0168 Room STA Byte.Counter.Lo
00AA- 86 28   0169      STX Byte.Counter.Mid
00AC- 84 29   0170      STY Byte.Counter.Hi
00AE- AD 54 C0 0171      LDA Page1        access main text screen
00B1- EE 00 04 0172      INC $400        show we received a byte
00B4- AD 55 C0 0173      LDA Page2        back to aux
00B7- AE A8 C0 0174      LDX Modem.ACIA.RxD
00BA- 9A      0175      TXS              pass it in S-reg
00BB- A6 21   0176      LDX Receive.Adr.Lo
00BD- A4 22   0177      LDY Receive.Adr.Hi
00BF- 2C 83 C0 0178      BIT LCRAM2        normally use LC bank 2
00C2- 98      0179      TYA
00C3- 29 F0   0180      AND #$F0
00C5- C9 C0   0181      CMP /$C000      if adr is in $CXXX range
00C7- D0 07   0182      BNE .1
00C9- 2C 8B C0 0183      BIT LCRAM1        use LC bank 1
00CC- 98      0184      TYA
00CD- 09 D0   0185      ORA /$D000
00CF- A8      0186      TAY
00D0- A5 23   0187 .1    LDA Receive.Bank
00D2- 4C 00 00 0188      JMP Write
00D5- E6 21   0189 W.Ret INC Receive.Adr.Lo fig next receive adr
00D7- D0 20   0190      BNE Scan.Printer.Port
00D9- E6 22   0191      INC Receive.Adr.Hi
00DB- D0 1C   0192      BNE Scan.Printer.Port
00DD- A5 23   0193      LDA Receive.Bank
00DF- C9 01   0194      CMP #1
00E1- 69 01   0195      ADC #1          clear carry if 0, else set it
00E3- C9 10   0196      CMP #$10
00E5- 90 08   0197      BCC .1          =>entering/still in Z-RAM
00E7- A9 00   0198      LDA #$00        wrap to //c bank 0
00E9- A2 7F   0199      LDX #IIC.Aux.Bank.Avail
00EB- A0 01   0200      LDY /IIC.Aux.Bank.Avail
00ED- 80 04   0201      BRA .2
00EF- A2 1E   0202 .1    LDX #Z.RAM.Banks.Avail
00F1- A0 00   0203      LDY /Z.RAM.Banks.Avail
00F3- 85 23   0204 .2    STA Receive.Bank
00F5- 86 21   0205      STX Receive.Adr.Lo
00F7- 84 22   0206      STY Receive.Adr.Hi
0207 Cant.Receive
0208 Scan.Printer.Port
00F9- A9 30   0209      LDA #$0011.0000 make transmit data reg empty and
00FB- 2D 99 C0 0210      AND Printer.ACIA.Status Data Carrier Detect mask
00FE- C9 10   0211      CMP #$0001.0000 test empty and DCD' lo
0100- D0 7A   0212      BNE Cant.Transmit =>not empty or not ready
0102- A5 27   0213      LDA Byte.Counter.Lo printer can take another byte,
0104- 05 28   0214      ORA Byte.Counter.Mid do we have one ?
0106- 05 29   0215      ORA Byte.Counter.Hi
0108- F0 72   0216      BEQ Cant.Transmit =>buffer is empty!!!
010A- A5 27   0217      LDA Byte.Counter.Lo do we release RTS' ?
010C- A6 28   0218      LDX Byte.Counter.Mid
010E- A4 29   0219      LDY Byte.Counter.Hi
0110- C9 91   0220      CMP #BufLen-2048
0112- D0 0D   0221      BNE .1          =>buffer not @ full-2048
0114- E0 F5   0222      CPX /BufLen-2048
0116- D0 09   0223      BNE .1          =>buffer not @ full-2048
0118- C0 08   0224      CPY BufLen-2048
011A- D0 05   0225      BNE .1          =>buffer not @ full-2048
011C- A5 2A   0226      LDA RTS.Bit
011E- 0C A4 C0 0227      TSB Modem.ACIA.Command release RTS' (maybe)
0121- 8D 78 C0 0228 .1    STA Z.RAM.Bank.Reg+5
0124- A5 27   0229      LDA Byte.Counter.Lo fig next byte count
0126- D0 08   0230      BNE .3

```

...Continued on page 16

Expanding Your IIC Is Easy With Z-RAM

Applied Engineering and Apple computer have teamed up to take your IIC to new heights.

Applied Engineering's Z-RAM card for the IIC is available with 256K or 512K of additional memory and a powerful Z-80 microprocessor for running CP/M software.

Z-RAM fits neatly inside the IIC. Installation is easy, clear instructions show you how. You'll need a screwdriver and about 10 minutes (if you can change a light bulb you can install Z-RAM).

Z-RAM and Appleworks will knock your socks off.



A 256K Z-RAM will give you a 229K available desktop and Appleworks will be completely loaded into memory. Appleworks will now run about 10 times faster in your IIC with 1 disk drive than in other IIC's with 2 disk drives. A 512K Z-RAM will give you a 413K available desktop. A 256K Z-RAM can be upgraded to 512K by just plugging in more memory chips.

Z-RAM is also a high speed solid state disk drive. With Z-RAM, your programs will load and save over 20 times faster. Z-RAM's RAM disk is compatible with Applesoft, ProDOS, DOS 3.3, PASCAL and CP/M. And with Z-RAM, you can copy a disk in one pass. Just insert the original, remove the original, insert blank disk! That's it! Z-RAM is another disk drive, only 20 times faster, 4 times larger capacity, and no whirring, clicking or waiting!

But before you start panting over all that extra memory, don't forget that the Z-RAM card has a built-in high speed Z-80 processor chip that allows you to run CP/M programs like Wordstar, dBASE II, Turbo PASCAL, Microsoft BASIC, FORTRAN and COBOL and over 3,000 other CP/M programs. So Z-RAM not only makes Apple programs run better and faster, it lets you run MORE programs.

With the Z-RAM card installed, your IIC is still your IIC only now you'll have that extra memory that Appleworks

and other programs need. And you can run all that great CP/M software that others can only dream about.

Z-RAM is 100% compatible with all IIC software and hardware including the mouse, 2nd disk, modem and printer. Z-RAM is easily handled by the IIC power supply as power consumption is kept very low by using two custom integrated circuits and a patent pending power saving design. And Z-RAM is from Applied Engineering, the acknowledged leader and innovator of accessories for the Apple.

Z-RAM comes complete with manual, RAM disk software, Z-80 operating system, CP/M manual and a 3 year no hassle warranty.

So the next time somebody asks you why you didn't get an IBM P.C., tell him you bought a IIC because the IBM didn't have enough memory and was too slow and couldn't run CP/M software. And tell him you made it past the 8th grade.

Z-RAM with 256K

\$449

Z-RAM with 512K

\$549

If you want to run CP/M software, but don't need more memory, may we suggest our Z-80c card. The Z-80c offers the same CP/M performance as Z-RAM but has no memory expansion ports. And the Z-80c will not affect the running of Apple programs. The Z-80c is priced at only \$159.00 and should you ever want to upgrade to Z-RAM, we'll refund your full purchase price.

Call (214) 241-6060

9 a.m. to 11 p.m. 7 days a week or

Send check or money order to:

Applied Engineering

P. O. Box 798

Carrollton, Texas 75006



MasterCard



Visa and

C.O.D. welcome. No extra charge for credit cards.

Texas residents add 5% sales tax. Add \$10.00 if outside U.S.A.



APPLIED ENGINEERING
"We Set the Standard"

How Many Bytes for each Opcode?.....Bob Sander-Cederlof

I have been thinking about a semi-automatic object code relocation scheme lately. Steve Wozniak wrote one for the 6502 back in 1976, published in various places such as Call APPLE's "Wozpak". But we are needing one for the 65C02, and maybe for the 65816.

Steve's version used his "Sweet-16" interpreter for some of the address arithmetic. That was okay, because Sweet-16 was in ROM in every Apple in those days. Not so now, although it is available to DOS 3.3 users as part of the Integer BASIC package. But we should write one that does not require Sweet-16.

Steve's relocater also used a ROM-based routine (part of the built-in disassembler) to determine how many bytes are used by each opcode. This routine has been modified in the //c monitor and the new enhanced //e monitor to include the 65C02 opcodes. That's nice, because that means Woz's program will automatically work with 65C02 programs if you run it with the new monitors. However, since I want to include all the 65816 opcodes, I need a new version.

The first step seems to be to write a program which will tell me how many bytes each opcode uses. I know that opcodes which are only one or two bytes do not need any relocation adjustments when a program is moved to a different place in memory. Most 3-byte and all 4-byte instructions contain absolute addresses; if an absolute address is inside the program being moved, it will have to be adjusted for the new location.

I haven't written the entire relocater yet, but I have written a program which will tell me all I need to know about the length of an opcode. My program returns the length in bytes and also two flags. One flag indicates the opcode is a 3-byte instruction which does include an absolute address. The other flag indicates the opcode was an immediate mode instruction. Immediate mode in 65816 code is ambiguous in length, except during execution. My program calls them two-byte instructions, but they may be three bytes each if the status bits so indicate at execution time. I am not sure how my relocater will handle this ambiguity, but for now I am content just to set a flag.

The code in the monitor which determines the length of opcodes uses a table lookup method. I figure that I could do that too, with a 64-byte table, using two bits for each opcode. I would still need a way to test for immediate mode and the special three-byte opcodes which do not have absolute addresses (MVP, MVN, PER, and BRL).

After looking at a chart which showed all the lengths, I decided to do it with bit analysis rather than table lookup. It is probably a little slower, but also a little smaller.

It turns out that almost all of the opcodes whose second hex digit is less than 8 use two bytes. There are only nine

exceptions. One interesting case here is BRK, which assembles to only one byte but is considered by the microprocessor to be a two-byte opcode. I am not sure whether the relocater should consider BRK as a single byte or a two-byte opcode, but I think it should probably be one byte.

All opcodes of the with the hex values of \$x8, \$xA, and \$xB are one byte, without exception. All opcodes with the hex values \$xC, \$xD, and \$xE are three bytes with absolute addresses, with only one exception: \$5C is a four-byte instruction. All opcodes with value \$xF are four bytes each.

The column of opcodes with values \$x9 are divided into two groups. Those with the first digit even (\$09, 29, 49, etc.) are all three bytes each with absolute addresses. The odd ones are immediate mode opcodes, which may be either two or three bytes each depending on status bits during execution.

Here is a table of the various byte counts, which was actually computed by my program. I printed "2#" for immediate mode opcodes, and "3+" for three-byte opcodes with absolute addresses.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	2	2	2	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
1	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
2	3+	2	4	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
3	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
4	1	2	2	2	3	2	2	2	1	2#	1	1	3+	3+	3+	4
5	2	2	2	2	3	2	2	2	1	3+	1	1	4	3+	3+	4
6	1	2	3	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
7	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
8	2	2	3	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
9	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
A	2	2	2	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
B	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
C	2	2	2	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
D	2	2	2	2	2	2	2	2	1	3+	1	1	3+	3+	3+	4
E	2	2	2	2	2	2	2	2	1	2#	1	1	3+	3+	3+	4
F	2	2	2	2	3+	2	2	2	1	3+	1	1	3+	3+	3+	4

The program which printed the table is in lines 1050-1320 below. The program which computes how many bytes in an opcode follows that. By inserting a "BEQ .6" between lines 1410 and 1420 I could make BRK a one-byte opcode.

My relocater should probably also be on the lookout for calls to ProDOS MLI. This is in effect a six-byte instruction. The first three bytes are \$20, \$00, \$BF (JSR MLI). The fourth byte is the MLI function code. The last two bytes are the address of a parameter table, and so should be considered as a relocatable address.

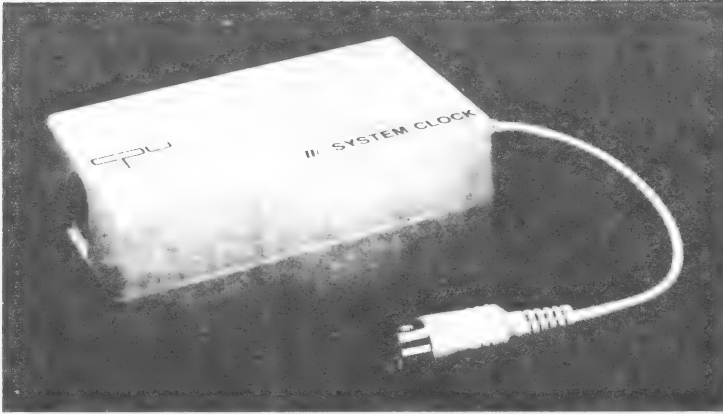
I hope to continue to pursue this idea of a relocater, but I make no promises. Maybe one of you would like to write one and share it with the rest of us.

```

1000 *SAVE S.BYTE TABLE
1010 *-----
FD8E- 1020 COUT .EQ $FDED
FD8E- 1030 CROUT .EQ $FD8E
1040 *-----
1050 T
0800- A2 00 1060 LDX #0
0802- 8A 1070 .1 TXA
0803- 29 0F 1080 AND #$0F
0805- D0 03 1090 BNE .2
0807- 20 8E FD 1100 JSR CROUT
080A- 8A 1110 .2 TXA
080B- 20 31 08 1120 JSR GET.LENGTH.OF.OPCODE
080E- 48 1130 PHA
080F- 29 07 1140 AND #$07
0811- 09 B0 1150 ORA #$07
0813- 20 ED FD 1160 JSR COUT
0816- 68 1170 PLA
0817- 0A 1180 ASL POSITION XY FOR INDEX
0818- 2A 1190 ROL
0819- 2A 1200 ROL
081A- 29 03 1210 AND #$03 0000 00XY
081C- A8 1220 TAY
081D- B9 2E 08 1230 LDA TABLE,Y
0820- 20 ED FD 1240 JSR COUT
0823- A9 A0 1250 LDA #""
0825- 20 ED FD 1260 JSR COUT
0828- E8 1270 INX
0829- D0 D7 1280 BNE .1
082B- 4C 8E FD 1290 JMP CROUT
1300 *-----
082E- A0 A3 AB 1310 TABLE .AS -/ #+/
1320 *-----
1330 * CALL WITH (A)= OPCODE
1340 * RETURN WITH (Y)= OPCODE
1350 * (A)= XY000LLL
1360 * LLL = # OF BYTES, 1...4
1370 * X = 1 IF ABS ADDRESS
1380 * Y = 1 IF IMMEDIATE
1390 *-----
1400 GET.LENGTH.OF.OPCODE
1410 TAY
0832- 29 0F 1420 AND #$0F
0834- C9 08 1430 CMP #$08
0836- 90 1E 1440 BCC .4 XXXX 0XXX
0838- C9 0C 1450 CMP #$0C
083A- 90 0E 1460 BCC .3 XXXX 10XX
083C- C9 0F 1470 CMP #$0F
083E- F0 07 1480 BEQ .2 XXXX 1111, L=4
0840- C0 5C 1490 CPY #$5C
0842- F0 03 1500 BEQ .2 0101 1100, L=4
1510 *---L=3. ABS ADDRESS---
0844- A9 83 1520 .1 LDA #$83
0846- 60 1530 RTS
1540 *---L=4---
0847- A9 04 1550 .2 LDA #4
0849- 60 1560 RTS
1570 *---XXXX 10XX---
084A- C9 09 1580 .3 CMP #$09
084C- D0 32 1590 BNE .6 X8, XA, or XB
1600 *---XXXX 1001---
084E- 98 1610 TYA
084F- 29 10 1620 AND #$10
0851- D0 F1 1630 BNE .1 XXX1 1001, L=3
1640 *---XX0 1001, IMMEDIATES, L=2---
0853- A9 42 1650 LDA #$42 OR 3 IF ## MODE
0855- 60 1660 RTS
1670 *---XXXX 0XXX---
0856- 4A 1680 .4 LSR CHECK ODD/EVEN
0857- B0 24 1690 BCS .5 ODD, L=2
0859- C0 22 1700 CPY #$22
085B- F0 EA 1710 BEQ .2 JSL LABS. L=4
085D- C0 20 1720 CPY #$20
085F- F0 E3 1730 BEQ .1 JSR ABS, L=3
0861- C0 40 1740 CPY #$40
0863- F0 1B 1750 BEQ .6 RTI, L=1
0865- C0 60 1760 CPY #$60
0867- F0 17 1770 BEQ .6 RTS, L=1

```

**NEW
PRODUCT**



IIC SYSTEM CLOCK

- Fully ProDos compatible
- Automatic time and date stamping
- Easy to use from BASIC
- Battery operated, uses 3 "AA" batteries (will last 1-2 years before simple replacement)
- Date has year, month, date and day of week
- Time has hours, minutes and seconds
- Will time and date stamp AppleWorks files
- Will display time and date on the AppleWorks screen
- Auto access from AppleWorks data-base (just use a time and date field)
- Pass through serial port - The IIC system clock can plug into either the modem or printer serial port, then modem or printer plugs into the clock
- No hassle 5 year warranty
- Only \$79.00



"We Set the Standard"

214-241-6060

APPLIED ENGINEERING

9 AM - 11 PM

```

0869- C0 62    1780    CPY #$62
086B- F0 16    1790    BEQ .7      PER LREL, L=3
086D- C0 82    1800    CPY #$82
086F- F0 12    1810    BEQ .7      BRL LREL, L=3
0871- C0 44    1820    CPY #$44
0873- F0 0E    1830    BEQ .7      MVP, L=3
0875- C0 54    1840    CPY #$54
0877- F0 0A    1850    BEQ .7      MVN, L=3
0879- C0 F4    1860    CPY #$F4
087B- F0 C7    1870    BEQ .1      PEA ABS, L=3
                1880    #---L=2-----
087D- A9 02    1890    .5    LDA #2      L=2
087F- 60        1900    RTS
                1910    #---L=1-----
0880- A9 01    1920    .6    LDA #1
0882- 60        1930    RTS
                1940    #---L=3. NON-ABS ADDRESS-----
0883- A9 03    1950    .7    LDA #3
0885- 60        1960    RTS
                1970    #-----

```

...Continued from page 10

```

0128- A5 28    0231    LDA Byte.Counter.Mid
012A- C0 02    0232    BNE .2
012C- D6 29    0233    DEC Byte.Counter.Hi
012E- C6 28    0234    .2    DEC Byte.Counter.Mid
0130- C6 27    0235    .3    DEC Byte.Counter.Lo
0132- AD 54 C0 0236    LDA Page1      access main text page
0135- EE 27 04 0237    INC $427      show we printed a byte
0138- AD 55 C0 0238    LDA Page2      back to aux
013B- A6 24    0239    LDX Transmit.Adr.Lo
013D- A4 25    0240    LDY Transmit.Adr.Hi
013F- 2C 83 C0 0241    BIT LCRAM2      normally use LC bank 2
0142- 98        0242    TYA
0143- 29 F0    0243    AND #$F0
0145- C9 C0    0244    CMP /$C000      if adr in $CXXX range
0147- D0 07    0245    BNE .4
0149- 2C 8B C0 0246    BIT LCRAM1      use LC bank 1
014C- 98        0247    TYA
014D- 09 D0    0248    ORA /$D000
014F- A8        0249    TAY
0150- A5 26    0250    .4    LDA Transmit.Bank
0152- 4C 11 00 0251    JMP Read
0155- 8D 98 C0 0252    R.Ret STA Printer.ACIA.TxD
0158- E6 24    0253    INC Transmit.Adr.Lo  fig next transmit adr
015A- D0 20    0254    BNE Next
015C- E6 25    0255    INC Transmit.Adr.Hi
015E- D0 1C    0256    BNE Next
0160- A5 26    0257    LDA Transmit.Bank
0162- C9 01    0258    CMP #1      clear carry if 0, else set it
0164- 69 01    0259    ADC #1
0166- C9 10    0260    CMP #$10
0168- 90 08    0261    BCC .1      =>entering/still in Z-RAM
016A- A9 00    0262    LDA #$00      wrap to //c bank 0
016C- A2 7F    0263    LDX #Iic.Aux.Bank.Avail
016E- A0 01    0264    LDY /Iic.Aux.Bank.Avail
0170- 80 04    0265    BRA .2
0172- A2 1E    0266    .1    LDX #Z.RAM.Banks.Avail
0174- A0 00    0267    LDY /Z.RAM.Banks.Avail
0176- 85 26    0268    .2    STA Transmit.Bank
0178- 86 24    0269    STX Transmit.Adr.Lo
017A- 84 25    0270    STY Transmit.Adr.Hi
                0271    Cant.Transmit
017C- 4C 2B 00 0272    Next JMP Scan
017F-          0273    Iic.Aux.Bank.Avail .eq #
08FD91-        0274    BufLen .eq $90000-Z.RAM.Used-Iic.Aux.Bank.Avail

```


I may have written hundreds of different versions of the elementary I/O conversion routines. The first few would have been for the IBM 704, back in college days. Then there was the G-15, the 1620, the 3100, the 3300, the 6600, the 1700, the 8090, the 960, the 980, the 990, and so on. Don't worry of those numbers don't mean anything to you. They are the "names" of computers out of the past, not micro chips.

What I am talking about is writing programs which convert input decimal characters representing decimal numbers into internal binary form, and the converse operation of converting binary numbers into decimal form. We have published several variations of both in previous newsletters, but I have some special ones to present here.

There are many variations of the basic routines, and that is one reason I have written so many. Thinking just of the output conversions (binary to decimal):

- * Convert to a string in memory, or print it out.
- * Number of bytes in binary number.
- * Supply leading zeroes or blanks or neither.
- * Integer, fraction, floating point, or fixed point.
- * Signed or unsigned.

The routine I set out to write today works with unsigned integers, prints out the resulting characters rather than storing them in a string, and does not print any leading zeroes or blanks. I wrote it to work with two-byte values, between 0 and 65535. As an added feature, I indicated in the comments how to expand it to work with larger values.

Lines 1800-2080 in the listing comprise the output conversion routine. I divide the number by ten, saving the remainder as the least significant digit; the quotient becomes the new number, so I repeat the process until the quotient is zero. Then the digits, which were all saved on the 6502 stack, are popped back off and printed.

Line 1810 starts the digit counter at 0, and line 1950 increments the counter each time a new digit is pushed onto the stack. Lines 2020-2060 pull the digits off the stack and print them in reverse order.

Lines 1970-2000 test the quotient: if it is non-zero, another division is performed; if not, we are ready to print the result. This is one place where you need to add code if your input values are larger than two bytes, as I indicated in line 1980. By the way, since we do one division before testing, an input value of zero will print as "0".

Lines 1830-1930 divide the input value by ten. It may look like I am dividing by five, but remember $5 = 10/2$. I did more fiddling than analyzing in this loop, but it really does work. Line 1840 sets the loop count to 16, the number of bits in two bytes. If you want to convert three-byte values, change the 16

to 24. The loop needs to be executed once for each bit in the input value. If you are going to have values longer than two bytes, you also need to add more ROL instructions between lines 1880 and 1900, as indicated in my comment line 1890. If you were to need a three byte conversion routine, you could just remove the "***" from the front of lines 1890 and 1980, and change line 1840 to LDY #24.

Notice that this subroutine is very short. and fairly fast. I have an idea that some of you will think of ways to make it shorter and faster; if you do, try to keep it easily modifiable for the number of bytes in values.

Next I wrote a program to convert from a decimal string into binary, lines 1290-1720. It is also set up for unsigned two-byte integer values, with comments indicating how to modify it for longer values. I have written shorter routines before, but this one makes extension to longer values easy and tests for overflow.

The string is assumed to be in ASCII, with high bits = 1, starting at \$0200, and terminated by any non-digit. It just so happens that these are just the conditions you usually find in an Apple, because almost all input routines use the buffer at \$0200. Woz started it, and we all followed Woz.

Lines 1300-1330 clear the value, as well as starting the buffer index at zero. The rest of the routine scans through the digits. Each time the current value is multiplied by ten, and the next digit added. If at any point an overflow is detected (a value too large for the number of bytes) the routine rings the bell and quits. You can use some other error indication, and probably should, such as printing "NUMBER TOO LARGE".

In order to multiply by ten, I set aside another storage area equal in length to the value accumulator. At line 1380 the new digit is saved in the Y-register. The accumulated value at this point is in XH and XL. Lines 1390-1480 form the value*4 in SH and XL, leaving the original value in XH and SL. (Yes, they are criss-crossed.) Lines 1410-1420 show how you would extend this portion to longer values.

Lines 1490-1610 add value*4 to value to get value*5, and then double the result to get value*10. Again, lines 1530-1550 show how to extend the value. Lines 1630-1700 add in the new digit, and the comments show how to extend to longer values.

The top level routine in lines 1130-1270 is just a test routine. It calls the monitor line input routine. If you type an empty line, it will stop. Otherwise it calls the input conversion routine, prints the resulting value in hexadecimal, and converts it back to decimal with the output conversion routine.



DISASM 2.2e - AN INTELLIGENT DISASSEMBLER : \$30.00

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new ASSEMBLY COOKBOOK. For entire Apple II family including the new Apple //c (with all the new opcodes). SOURCE CODE available for an additional \$30.00

LOW LOW PRICE !!! C-PRINT For The APPLE //c : \$69.00

Connect standard parallel printers to an Apple //c. C-PRINT is a hardware accessory that plugs into the standard Apple //c printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

FONT DOWNLOADER & EDITOR : \$39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, //e. Specify printer: Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

The Font Downloader & Editor for the Apple Imagewriter Printer. For use with Apple II, II+, //e (with SuperSerial card) and the new Apple //c (with builtin serial interface).

FONT LIBRARY DISKETTE #1 : \$19.00 Contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

The 'PERFORMER' CARD : \$39.00

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. SOURCE CODE: \$30.00

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM : \$25.00

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands. SOURCE CODE: \$50.00

RAM/ROM DEVELOPMENT BOARD : \$30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Cn00-CnFF and \$C800-CFFF.

ALL NEW !!! MIDI MUSIC PRODUCTS

MIDI means Musical Instrument Digital Interface. Use your computer with any MIDI-equipped music keyboard for entertainment and music education. Low cost MIDI player interface cable, complete with 6 song demo disk: \$49.00. Thousands of popular songs available soon on diskette (also compatible with Passport MIDI interface). Products for both the Apple IIc and Commodore 64/128. Unique general purpose MIDI expander cable and gender changer also available. Send SASE for product descriptions and prices.

Avoid a \$3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders accepted. RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885



```

1000 *SAVE S.BINDEC
1010 *-----
00- 1020 XL .EQ $00
01- 1030 XH .EQ $01
10- 1040 SL .EQ $10
11- 1050 SH .EQ $11
1060 *-----
FBDD- 1070 BELL .EQ $FBDD
FD6A- 1080 RDLINE .EQ $FD6A
FD8E- 1090 PRBYTE .EQ $FD8E
1100 COUT .EQ $FDED
1110 CROUT .EQ $FD8E
1120 *-----
1130 T
0800- 20 6A FD 1140 JSR RDLINE
0803- 8A 1150 TXA
0804- D0 01 1160 BNE .1
0806- 60 1170 RTS
0807- 20 22 08 1180 .1 JSR CONVERT.DEC.TO.BIN
080A- A5 01 1190 LDA XH
080C- 20 DA FD 1200 JSR PRBYTE
080F- A5 00 1210 LDA XL
0811- 20 DA FD 1220 JSR PRBYTE
0814- A9 BD 1230 LDA # "="
0816- 20 ED FD 1240 JSR COUT
0819- 20 6C 08 1250 JSR CONVERT.BIN.TO.DEC
081C- 20 8E FD 1260 JSR CROUT
081F- 4C 00 08 1270 JMP T
1280 *-----
1290 CONVERT.DEC.TO.BIN
0822- A2 00 1300 LDX #0
0824- 86 00 1310 STX XL least significant byte
1320 *-- STX XI ---ANY INTERMEDIATE BYTES---
0826- 86 01 1330 STX XH most significant byte
0828- BD 00 02 1340 .1 LDA $200,X
082B- 49 B0 1350 EOR #0
082D- C9 0A 1360 CMP #10
082F- B0 36 1370 BCS .3 ...END OF NUMBER
0831- A8 1380 TAY SAVE CURRENT DIGIT
0832- A5 00 1390 LDA XL
0834- 85 10 1400 STA SL
1410 *-- LDA XI ---ANY INTERMEDIATE BYTES---
1420 *-- STA SI ---FOLLOW THIS PATTERN-----
0836- A5 01 1430 LDA XH
0838- 20 68 08 1440 JSR SHIFT.X
083B- B0 27 1450 BCS .2 ...OVERFLOW
083D- 20 68 08 1460 JSR SHIFT.X
0840- B0 22 1470 BCS .2 ...OVERFLOW
0842- 85 11 1480 STA SH
0844- 18 1490 CLC
0845- A5 00 1500 LDA XL
0847- 65 10 1510 ADC SL
0849- 85 00 1520 STA XL
1530 *-- LDA XI ---ANY INTERMEDIATE BYTES---
1540 *-- ADC SI ---FOLLOW THIS PATTERN-----
1550 *-- STA XI -----
084B- A5 01 1560 LDA XH
084D- 65 11 1570 ADC SH
084F- B0 13 1580 BCS .2 ...OVERFLOW
0851- 20 68 08 1590 JSR SHIFT.X
0854- B0 0E 1600 BCS .2 ...OVERFLOW
0856- 85 01 1610 STA XH
0858- E8 1620 INX SCAN TO NEXT DIGIT
0859- 98 1630 TYA GET DIGIT
085A- 65 00 1640 ADC XL LEAST SIGNIFICANT BYTE
085C- 85 00 1650 STA XL
085E- 90 C8 1660 BCC .1 ...NO CARRY
1670 *-- INC XI ---ANY INTERMEDIATE BYTES---
1680 *-- BNE .1 ---FOLLOW THIS PATTERN-----
0860- E6 01 1690 INC XH MOST SIGNIFICANT BYTE
0862- D0 C4 1700 BNE .1 ...UNLESS OVERFLOW
0864- 20 DD FB 1710 .2 JSR BELL SIGNAL OVERFLOW
0867- 60 1720 .3 RTS
1730 *-----

```

```

0868- 06 00      1740 SHIFT.X
                  1750 ASL XL      LEAST SIGNIFICANT BYTE
086A- 2A         1760 #-- ROL XI    ---ANY INTERMEDIATE BYTES---
086B- 60         1770 ROL      ...MOST SIGNIFICANT BYTE IN A
                  1780 RTS
                  1790 #-----
086C- A2 00      1800 CONVERT.BIN.TO.DEC
                  1810 LDX #0      DIGIT COUNTER
086E- A9 00      1820 #---DIVIDE BY TEN-----
0870- A0 10      1830 .1 LDA #0
0872- C9 05      1840 LDY #16    2*(# Bytes being converted)
0874- 90 02      1850 .2 CMP #5
0876- E9 05      1860 BCC .3
0878- 26 00      1870 SBC #5
                  1880 .3 ROL XL
087A- 26 01      1890 #-- ROL XI    ---ANY INTERMEDIATE BYTES---
087C- 2A         1900 ROL XH
087D- 88         1910 ROL
087E- D0 F2      1920 DEY
0880- 48         1930 BNE .2
0881- E8         1940 PHA      SAVE DIGIT ON STACK
                  1950 INX      COUNT THE DIGIT
0882- A5 00      1960 #---NEXT DIGIT-----
0884- 05 01      1970 LDA XL
0886- D0 E6      1980 #-- ORA XI    ---ANY INTERMEDIATE BYTES---
                  1990 ORA XH
0888- 68         2000 BNE .1
0889- 09 B0      2010 #---PRINT DECIMAL-----
088B- 20 ED FD   2020 .4 PLA
088E- CA         2030 ORA #0
088F- D0 F7      2040 JSR COUNT
0891- 60         2050 DEX
                  2060 BNE .4
                  2070 RTS
                  2080 #-----

```

8086/8088 Cross Assembler

Use your Apple to learn 8086 programming! You can program for the IBM PC, the clones, and ALF's co-processor board without ever leaving the friendly environment of Apple DOS 3.3.

This easy-to-use cross assembler, based on the S-C Assembler II (Version 4.0), covers all the 8086 and 8088 instructions and all the addressing modes. Instruction mnemonics are based on the Microsoft 8086 assembler. Does not include newer S-C Assembler features like macros or the EDIT command.

Documentation covers the differences from standard S-C Assembler operation and syntax. Sample source programs help you become familiar with the assembler syntax.

With permission from S-C Software, XSM 8086/8088 is available to owners of any S-C Assembler for \$80.00 post-paid. (No credit cards or purchase orders.)

Don Rindsberg
The Bit Stop
5958 S. Shenandoah Rd.
Mobile, AL 36608

(205) 342-1653

Over the years I have fallen into certain habits when it comes to naming files. I find it convenient to use names starting with "S." for assembly language source files, "B." for binary object code files, and so on. Others like to use suffixes like ".SRC" and ".OBJ" for the same reasons. Some operating systems, like CP/M for example, use suffixes to indicate file type. Others, like ProDOS, let you build sub-directories to categorize your files.

Sometimes I would like to have the ability to do the same operation on a whole group of files. For example, I might want to DELETE all files starting with "B.". Or I might want to copy a whole group of files from one disk to another. If the files happen to have similar names, and if DOS allowed wildcards in filenames, it would be easier.

Some DOS 3.3 programs do have this feature: Apple's FID program, Sensible Software's Super Disk Copy, and others. They have a method for specifying a filename without spelling out the entire name.

The subroutine inside DOS 3.3 which compares a filename you have specified with the names in a catalog is found at \$B1F5:

```

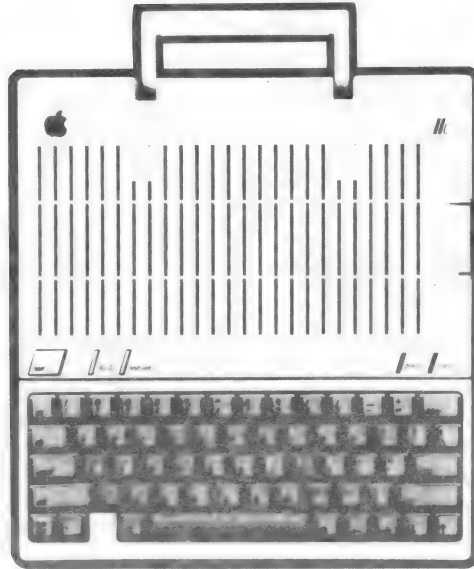
        LDY #0
        INX
        INX
.1      INX
        LDA ($42),Y   Filename you specified
        CMP $B4C6,X   Filename in catalog sector
        BNE ...       ...did not match
        INY
        CPY #30
        BNE .1
        ... matched ...
    
```

This is a very straightforward string comparison. It requires an exact match of all 30 characters of a filename. There is a similar routine at \$A782 which compares a filename you specify with the filenames in the open file buffers.

I wrote a subroutine called MATCH which compares two 30-character strings, allowing wildcards. Unfortunately, it not a simple matter to plug such a subroutine into DOS 3.3, and I have not done that. It is more likely that this subroutine will find its way into some future utility programs.

I also wrote a testing program, so that I could see if my code worked. The program in lines 1110-1380 searches through a list of 30-character strings, printing those which match a key string. To simplify my test program (a good idea to keep testers simple, so they are not themselves more buggy than the testees!) I assembled in the key string and the list of strings to be searched. A slightly better test would allow me to type in the key string.

640K



Fatter than a Fat Mac and ready for serious business — that's an Apple //c with a MultiRam™ C Card installed. MultiRam adds from 256K to 512K to your //c's own 128K memory to give you up to a 640K //c.

With a 640K Apple //c you can...

- Run AppleWorks with a 413K Desktop. That's a Desktop large enough for the most demanding business or personal use. Data bases and spreadsheets can grow beyond the frustrating limitations of the old 55K Desktop.

You can even eliminate AppleWork's need for an external disk drive and speed up AppleWorks by loading all the program disk into MultiRam's memory.

- Run the new SuperCalc3[®], an integrated spreadsheet for your //c with the power of Lotus 1-2-3, and create complex models up to 512K in size.
- Run FlashCalc, Magic Office, MagicCalc and future programs that will use most or all of MultiRam's memory.

Apple is the registered trademark of Apple Computer, Inc. AppleWorks and Macintosh are trademarks of Apple Computer, Inc. FlashCalc is the trademark of Paladin Software Corporation. Lotus 1-2-3 is the trademark of Lotus Development Corporation. Magic Office and MagicCalc are trademarks of Artsci, Inc. SuperCalc[®] is the trademark of Sorcim/IUS, Inc.



- Use MultiRam C's memory as a RAM disk for ProDOS and DOS equaling up to four standard //c disk drives with speeds more than 20 times normal.

The MultiRam C Card...

- Easily installs inside the //c using only a screwdriver. No messy soldering or jumpers are required. MultiRam's expandable memory is easily increased if you choose a 256K version.
- Includes a comprehensive manual, AppleWorks memory expansion software, ProDOS and DOS RAM disk software, and RAM test software at no additional charge.
- Features a full 5-year warranty and is 100% compatible with all //c hardware and software. MultiRam uses a low 1.2 watts of power with its CMOS power saving design. CMOS insures longer sessions when using a battery pack with your //c.

Available at a **DISCOUNT** from:

Coit Valley Computers
14055 Waterfall Way, Dallas, Texas 75240
214-234-5047

Checkmate Technology, Inc.

509 South Rockford Drive • Tempe, Arizona 85281-3021
(602) 966-5802 U.S.A. Telex: 165-025 CEC PHX

My MATCH program assumes that the address of the string to be compared with the key is stored at FN and FN+1. Characters in the filename are addressed by "(FN),Y", and in the key are addressed by "KEY,X". MATCH will return with carry set if the filename matches the key, and carry clear if not.

Both the filename and the key are stored "left-justified, blank-filled". That means there may be any number of non-significant blanks on the right end. Lines 1490-1530 scan the current filename from right-to-left, looking for the last non-blank in the name. Lines 1550-1590 do the same for the key. If there is any chance either filename or key could be completely blank, an extra line "BMI ERROR" should be inserted at 1505 and 1565.

I save the index to the right end of the key in KEY.START. Because the end of the filename and key strings is variable, I actually do the comparison from right to left. This makes the "end" actually the beginning.

Line 1610 could be "JMP .4" or "BNE .4", because the object is to get to line 1660. However, the "INX" allows me to fall through lines 1630-1640 and it takes only one byte rather than two or three.

The comparison begins at line 1660. Remember we are scanning backwards, from right to left. Lines 1660-1670 save the two string pointers. Line 1680 gets the next character from the key. If it is a wildcard, I branch back to line 1630. Note that all that happens is that the wildcard is skipped over!

If the key character is not a wildcard, it gets compared to the next character of the filename at line 1710. If it matches, lines 1730-1760 advance both pointers and the comparison continues. These lines also check to see if we have come to the left end of the key or of the filename.

If we are at the end of the filename, lines 1770-1820 check the rest of the key. If there are any characters left in the key which are not wildcards, then the current filename does not match. Otherwise, it does match. Lines 1830-1880 set the appropriate carry status and return.

If we are at the end of the key, lines 1900-1910 check whether we are also at the end of the filename. If so, the filename matched. If not, maybe it did not match. I say maybe, because if there was a wildcard, we might come out with a match if we widen the amount matched by that wildcard. Lines 1920-1990 will handle that possibility.

Two conditions bring us to line 1930. Either a character in the key did not match the current character in the filename, or there are unmatched filename characters left over after the end of the key. In either case, if there has been no wildcard in the key (so far), then the filename does not match the key. If there has been a wildcard, we can try again to match from the most recent wildcard on. We can tell whether or not there has been a wildcard so far by comparing KEY.PNTR with KEY.START.

12 Good Reasons Why RAMWORKS™ Is The Best Expansion Card For Your IIfx

1 APPLEWORKS MEMORY Even though Ramworks enhances and expands a VAST ARRAY of other programs, Appleworks is our claim to fame. A 64K Ramworks will ADD +6K to your available desktop memory, a 128K Ramworks will ADD 91K, a 256K Ramworks will ADD 182K, and a 512K Ramworks will ADD 364K and a 1 meg Ramworks will give you nearly an 800K desktop. And it's all done automatically! When you plug in more memory chips into your Ramworks card, Appleworks will find them automatically. Ramworks also increases the maximum number of records from 1350 to 4300.

2 APPLEWORKS SPEED AND POWER Ramworks does more than just increase the desktop memory (as if that weren't enough). With Ramworks, Appleworks will be able to run up to 20 times faster. If you buy a 256K or larger Ramworks card, Appleworks will automatically load itself in Ramworks. This greatly increases the speed at which Appleworks operates by eliminating all that nasty, time consuming disk access on Drive 1. These are but a few reasons why we say that Ramworks is Appleworks best friend.

3 EXPANDABILITY Ramworks was designed with the future in mind, as your needs increase, so can Ramworks. Clear instructions show you how to plug in more memory (up to 1 meg).

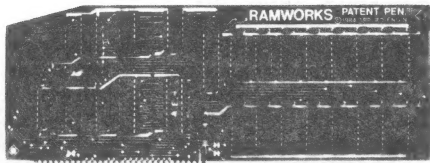
4 SPEED Today, as programs become more and more sophisticated, they inevitably become larger. And many of today's best selling programs (like Appleworks) won't fit in a 128K Apple, so many of these new larger programs continually go back to disk in search of more data. With Ramworks, you can have enough memory so that the entire program will be loaded into Ramworks' memory. This greatly increases the speed of software because your disk runs at 300 RPM, but Ramworks operates at the speed of light!

5 COLOR The same slot that's used for memory expansion is also the slot that's used for RGB color display, so all those lesser memory cards of yesterday make you decide in advance if you want RGB color. Only Ramworks lets you decide later to add RGB color. For only \$129 an RGB option can be added to Ramworks to give you double high resolution color graphics and 80 column text. All with a razor sharp, vivid brilliance that's unsurpassed in the industry. The RGB option does not waste another valuable slot, but rather plugs into the back of Ramworks and attaches to any Apple compatible monitor. Remember, you can order the RGB option with your Ramworks or add it on at a later date.

6 COMPATIBILITY, OF THE SOFTWARE KIND Programs like Appleworks, Magic Office System, Flashcalc, The Spread Sheet, Diverse A-Dos, Supercalc, Magicle, and many others automatically recognize all or most of Ramworks memory (512K is average). The simple fact is that Ramworks is compatible with more off-the-shelf software than any other RAM card. Ramworks is 100% compatible with ALL software written for the Apple 80 column and extended 80 column card. Additionally, Ramworks can emulate other RAM cards so software written for other cards will run without modification. Software written for RAMWORKS will not work on other cards. We can emulate others, but others can't emulate us.

7 COMPATIBILITY, OF THE HARDWARE KIND Unlike others, Ramworks is fully compatible with hardware add on's from other companies, like the Sider and Profile hard disks. And Ramworks was designed in accordance with the official expansion rules defined by Apple so you don't have to worry about compatibility problems. As you continue to expand and make your Apple more powerful with other expansion products from Applied Engineering, you'll appreciate how each product has extra features designed to work with Ramworks and other products to give you a total performance package that is more powerful than the sum of its parts.

8 IT SELLS THE MOST Popularity translates into great software support because software companies can't support all RAM cards, they can only support the ones their customers are likely to own. And software companies appreciate the fact that when they write software for Ramworks in the IIfx,



they're also writing software for our memory expansion card for the IIfx, Z-RAM. And our customer list reads like the Who's Who of Apple computing with just about every software company in the land buying one, including Apple Computer (in the hundreds), Rupert Lissner, and Steve Wozniak (we didn't give one to Mr. Wozniak just to use his name, 2 one meg Ramworks were paid for at full price).

9 IT'S FROM APPLIED ENGINEERING Unlike most of the competition, we only make accessories for Apple, so we'll never spend your money on IBM product research. Applied Engineering's years of experience and wide product line really pays off, and because of our high sales levels we buy most of our I.C. chips factory direct. So don't let our low prices fool you, they're caused by high volume production. That's why we can offer the most memory for the least money. Guaranteed!

10 IT'S GOT IT ALL

- ✓ Sharp 80 Column Text
- ✓ Double high resolution graphics (with or without RGB option)
- ✓ User Expandable to 1 Megabyte
- ✓ Can Use 64K or 256K RAMs in any combination
- ✓ Adds Memory to Appleworks
- ✓ Accelerates Appleworks
- ✓ 100% Compatibility with All IIfx software
- ✓ RAM Disk software available, compatible with Applesoft, PRO-DOS, DOS 3.3, and PASCAL (\$29)
- ✓ RAM Disk available for CP/M (\$29). (This program is included with our CP/M card)
- ✓ Visicalc preboot available (\$29)
- ✓ RGB option
- ✓ Takes only one slot
- ✓ 3 year no hassle warranty

11 THE PATENT OFFICE HAS ONE There are many advanced features on Ramworks, but two parts of the design are so advanced we applied for patents. One patent application deals with our ultra fast, ultra smooth 80 column screen display, and the other patent application deals with our ingenious way of dramatically reducing the power and heat of memory chips and improving reliability at the same time.

12 HERE TODAY, HERE TOMORROW In the seven years we've been making products for the Apple, we've seen a lot of companies come and go. Although nothing is forever, we're growing, expanding and we're profitable. And we are totally committed to Apple computing, which means you'll never run out of things to do with Ramworks. Or for that matter, reasons to buy one.

Ramworks™ with 64K	\$179
Ramworks™ with 128K	\$249
Ramworks™ with 256K	\$299
Ramworks™ with 512K	\$399
Ramworks™ with 1 MEG	\$649
RGB Option (can be added later)	\$129

Call (214) 241-6060

9 a.m. to 11 p.m. 7 days a week or send check or money order to: Applied Engineering, P. O. Box 798, Carrollton, Texas 75006



MasterCard Visa and

C.O.D. welcome. No extra charge for credit cards. Texas Residents add 5% sales tax. Add \$10.00 if outside U.S.A.



APPLIED ENGINEERING
"We Set the Standard"

If they are the same, there has been no wildcard. Lines 1920-1990 handle all these details.

I made the wild card character itself a variable, so that you could change it by program control. Since "=" is a valid character in a filename, you might want to use something else.

With this kind of MATCH subroutine, a key of "=".OBJ" would match all names ending with ".OBJ"; "S.=" would match all names starting with "S."; "=A=B=" would match all names containing "A" followed by "B".

You can see the similarity between MATCH and a global search capability such as you might find in a word processor, or in the S-C Macro Assembler. The FIND and REPLACE commands in S-C Macro allow wildcards. However, MATCH differs in that it anchors the key to the beginning and end of the file name (unless you specify a wildcard in those positions).

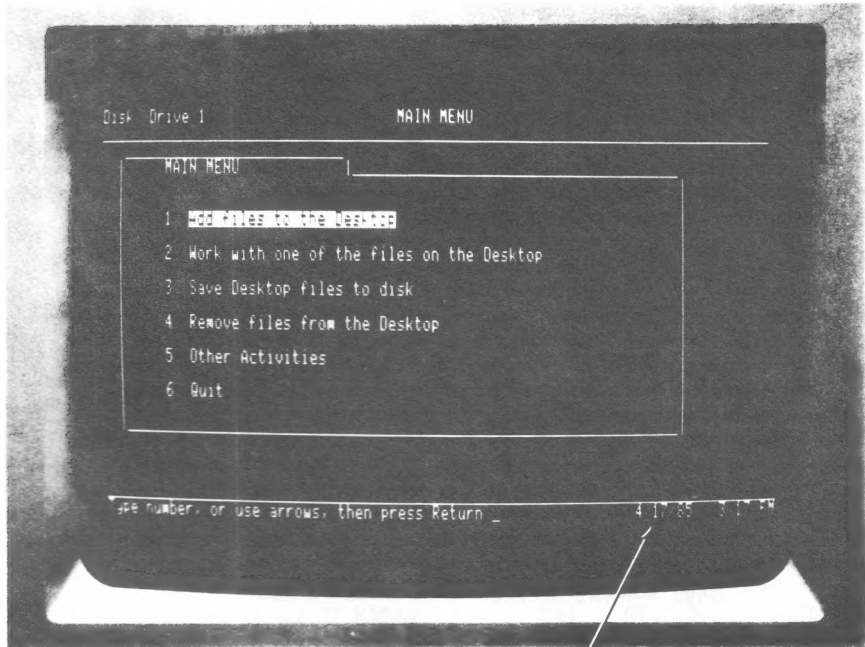
If string comparisons of this type intrigue you, the book "Software Tools" develops an extremely powerful one in chapter 5. "Software Tools" is a classic book by Kernighan and Plauger, available at many bookstores. (A "classic" in computer books is one still in print after five years; this one qualifies, since it was originally published in 1976.) Their string match routine allows single- and multi-character wildcards, semi-wildcards that match subsets of characters, control of anchoring, and more. It would be a worthwhile exercise to try implementing their algorithm in 6502 language.

```

1000 *SAVE S.WILDCARD
1010 *-----
1020 COUT .EQ $FDED
1030 CROUT .EQ $FD8E
1040 *-----
00- 1050 KEY.PNTR .EQ $00
01- 1060 BUF.PNTR .EQ $01
02- 1070 FN .EQ $02,03
04- 1080 KEY.START .EQ $04
05- 1090 CNTR .EQ $05
1100 *-----
1110 T
0800- A9 04 1120 LDA #NAME.CNT
0802- 85 05 1130 STA CNTR
0804- A9 A1 1140 LDA #FNLIST
0806- A0 08 1150 LDY /FNLIST
0808- 85 02 1160 .1 STA FN
080A- 84 03 1170 STY FN+1
080C- 20 32 08 1180 JSR MATCH
080F- 90 03 1190 BCC .2 ...DID NOT MATCH
0811- 20 23 08 1200 JSR DISPLAY
0814- A5 02 1210 .2 LDA FN
0816- 18 1220 CLC
0817- 69 1E 1230 ADC #30
0819- A4 03 1240 LDY FN+1
081B- 90 01 1250 BCC .3
081D- C8 1260 INY
081E- C6 05 1270 .3 DEC CNTR
0820- D0 E6 1280 BNE .1
0822- 60 1290 RTS
1300 *-----
1310 DISPLAY
0823- A0 00 1320 LDY #0
0825- B1 02 1330 .1 LDA (FN),Y
0827- 20 ED FD 1340 JSR COUT
082A- C8 1350 INY
082B- C0 1E 1360 CPY #30
082D- 90 F6 1370 BCC .1
082F- 4C 8E FD 1380 JMP CROUT

```

If You Have
APPLEWORKS™
 It's Easy To Tell If You
 Have A Timemaster H.O. Clock
 In Your Apple



Just Look Right Here

Only the Timemaster H.O. displays the date and time on the Appleworks screen.* If you don't have a Timemaster H.O., you'll just get the help key reminder. The Timemaster H.O. will also automatically time and date stamp your files on disk. And don't forget, the Timemaster H.O. has all the features of all the competition combined, including year, leap year (not just in PRO-DOS), month, date, day, hours, minutes, seconds and milliseconds. The Timemaster H.O. is compatible with PRO-DOS, DOS 3.3, PASCAL and CP/M. And the Timemaster H.O. automatically emulates all other clock cards so you won't have any compatibility problems because the Timemaster H.O. works with ANY program that reads ANY clock.

In fact, you could put ALL the competitive cards in every slot in your Apple and you still wouldn't have all the features of the Timemaster H.O.

The Timemaster H.O. comes with a ton of fun and useful software. It has an easy to read yet detailed manual, a 20 year auto-recharging battery and a 3 year no hassle warranty.

TIMEMASTER H.O.

SIMPLY PUT,

IT'S SIMPLY THE BEST

\$129.00 Complete

*If you purchased a Timemaster H.O. prior to AppleWorks support, an easy to use patch program is available for \$20.00.



Call (214) 241-6060 9 a.m. to 11 p.m., 7 days a week or
 Send check or money order
 P. O. Box 798
 Carrollton, Texas 75006



MasterCard, Visa and C.O.D. welcome. No extra charge for credit cards.

Texas residents add 5%% sales tax.

Add \$10.00 if outside U.S.A.

```

1390 *-----
1400 *   COMPARE KEY TO A FILE NAME
1410 *   KEY MAY CONTAIN WILDCARDS
1420 *   TRAILING BLANKS DON'T COUNT
1430 *   FILE NAME ADDRESSED VIA "(FN),Y"
1440 *   KEY ADDRESSED VIA "KEY,X"
1450 *   KEY AND FILE NAME ARE UP TO 30 CHARS LONG
1460 *   (STORED LEFT-JUSTIFIED, BLANK-FILLED)
1470 *-----
1480 MATCH
0832- A0 1E 1490 LDY #30 FIND LAST NON-BLANK CHAR
0834- 88 1500 DEY IN FILE NAME
0835- B1 02 1510 LDA (FN),Y
0837- C9 A0 1520 CMP #""
0839- F0 F9 1530 BEQ .1
1540 *-----
083B- A2 1E 1550 LDX #30 FIND LAST NON-BLANK CHAR
083D- CA 1560 DEX IN KEY
083E- BD 81 08 1570 LDA KEY,X
0841- C9 A0 1580 CMP #""
0843- F0 F8 1590 BEQ .2
0845- 86 04 1600 STX KEY.START
0847- E8 1610 INX
1620 *---WILD CARD---
0848- CA 1630 .3 DEX ADVANCE KEY POINTER
0849- 30 21 1640 BMI .8 ...END OF KEY IS WILD, SO MATCHES
1650 *-----
084B- 86 00 1660 .4 STX KEY.PNTR
084D- 84 01 1670 .5 STY BUF.PNTR
084F- BD 81 08 1680 .6 LDA KEY,X
0852- CD 80 08 1690 CMP WILD.CARD
0855- F0 F1 1700 BEQ .3 ...WILD CARD CHARACTER
0857- D1 02 1710 CMP (FN),Y
0859- D0 18 1720 BNE .11 ...NO MATCH
085B- CA 1730 DEX
085C- 30 12 1740 BMI .10 ...END OF KEY
085E- 88 1750 DEY
085F- 10 EE 1760 BPL .6 ...STILL MORE TO COMPARE
1770 *---END OF FILE NAME, MORE KEY---
0861- BD 81 08 1780 .7 LDA KEY,X
0864- CD 80 08 1790 CMP WILD.CARD
0867- D0 05 1800 BNE .9 ...REST OF KEY NOT WILD, NO MATCH
0869- CA 1810 DEX
086A- 10 F5 1820 BPL .7
1830 *---VALID MATCH---
086C- 38 1840 .8 SEC SIGNAL MATCH
086D- 60 1850 RTS
1860 *---NOT A MATCH---
086E- 18 1870 .9 CLC
086F- 60 1880 RTS
1890 *---END OF KEY---
0870- 88 1900 .10 DEY MATCH IF END OF NAME
0871- 30 F9 1910 BMI .8 ...END OF NAME
1920 *---IF AFTER WILDCARD, SLIP---
0873- A6 00 1930 .11 LDX KEY.PNTR START KEY OVER AGAIN
0875- E4 04 1940 CPX KEY.START
0877- F0 F5 1950 BEQ .9 ...NOT AFTER A WILDCARD
0879- A4 01 1960 LDY BUF.PNTR SLIP TO LEFT IN BUFFER
087B- 88 1970 DEY
087C- 10 CF 1980 BPL .5 TRY AGAIN
087E- 30 E1 1990 BMI .7 REST OF KEY BETTER BE WILD
2000 *-----
0880- BD 2010 WILD.CARD .AS -/=/
2020 *-----
2030 KEY .AS -/A= /
2040 *-----
2050 FNLIST .AS -/A SIMPLE KEY /
2060 .AS -/NOT SUCH A SIMPLE KEY /
2070 .AS -/NOT A SIMPLE KEY AT ALL /
2080 .AS -/A SIMPLE KEY AFTER ALL /
2090 NAME.CNT .EQ #-FNLIST/30
2100 *-----

```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$14 postage for other countries. Back issues are available for \$1.80 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)